NISTIR 5985

A Fortran 90 Interface for OpenGL

William F. Mitchell

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Information Technology Laboratory
Gaithersburg, MD 20899-0001

NIST

NISTIR 5985

A Fortran 90 Interface for OpenGL

William F. Mitchell

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Information Technology Laboratory
Gaithersburg, MD 20899-0001

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

A Fortran 90 Interface for OpenGL

William F. Mitchell*
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899
william.mitchell@nist.gov

Abstract

It is important to provide a good fortran interface to OpenGL and related libraries for scientific visualization in mathematical software. OpenGL currently provides a fortran interface which can be used by fortran 77 or fortran 90 programs. However, this interface relies upon several extensions to the fortran 77 standard. By using the new features of fortran 90 it is possible to define an interface to OpenGL that does not depend on any extensions to the standard and provides access to the full functionality of OpenGL. This document defines such an interface.

1 Introduction

Most mathematical software for scientific computing is written in fortran, and most scientific computing applications require 3D graphics for visualization. It is therefore important to provide a good fortran interface to OpenGL and related libraries. OpenGL [3] currently provides a fortran interface [1] which can be used by fortran 77 or fortran 90 programs. However, this interface relies upon several extensions to the fortran 77 standard. Although some of these extensions are commonly used by fortran compilers (e.g., `real*4`, `real*8`, `integer*4`) and some have been made standard in fortran 90 [2] (e.g., `include`, identifiers up to 31 characters, underscore character in identifiers), others are not widely supported (e.g., `logical*1`, `integer*1`, `integer*2`, identifiers longer that 31 characters), which makes OpenGL difficult or impossible to use from some fortran processors. Also, some of the OpenGL functionality cannot be achieved by any fortran processor under the current fortran binding (e.g., arbitrary length character string function result).

By using the new features of fortran 90 it is possible to define an interface to OpenGL that does not depend on any extensions to the standard and provides access to the full functionality of OpenGL. It can also increase the capability of robustness and portability in the user application code, and increase the similarity between the fortran and C interfaces.

This document defines a fortran 90 interface for OpenGL. It is not intended to replace the existing fortran interface (henceforth referred to as the fortran 77 interface) at this time, since the existing interface will be required on systems that are still using a fortran 77 compiler. The fortran 90 interface is intended to provide an alternative through which the fortran 90 programmer can achieve robustness and portability in an OpenGL application program. A reference implementation of the fortran 90 interface has been made available to the public at http://math.nist.gov/f90gl.

The major differences between the fortran 77 and fortran 90 interfaces are:

- The interface is accessed through modules, rather than include statements. Among other advantages of modules, this provides explicit interfaces to the OpenGL procedures for improved robustness.
- Kind type parameters are provided for matching fortran types to C types. This eliminates the need for nonstandard "*byte" declarations. It also provides a mechanism for transparently handling type mismatches on systems in which the fortran processor does not support all the C types used by OpenGL, for increased portability.
- Fortran derived types are provided where C structs are used in the interface. This increases the similarity between the fortran and C interfaces, and provides a mechanism through which the implementor can encapsulate whatever interface data is required.
- The fortran functions corresponding to C functions that return a pointer to a character string now return a pointer to an array of characters. This increases the similarity between the fortran and C interfaces, and adds the capability of arbitrary length character string return values.
- Extremely long names are truncated to 31 characters to comply with the fortran 90 standard, and the prefix is changed to f90gl to avoid name space clashes with the OpenGL library and fortran 77 interface.

This interface explicitly covers the OpenGL 1.1 core library, and the GLU library. The principles laid out in this interface can also be applied to related libraries, toolkits, and OpenGL extensions. Some entities from the OpenGL tk toolkit and the Graphics Library Utility Toolkit (GLUT) are used for illustration in this document.

2 Interface Definition

This section describes and discusses the fortran 90 interface to OpenGL.

2.1 Modules

The fortran 90 interface to OpenGL is accessed through modules. The modules provide access to kind type parameters, defined constants, procedures, and derived types (structures).

The module f90gl_kinds contains the definitions of the kind type parameters as described in Section 2.2. This module is not normally used directly in application code, but is inherited through the other modules. The kind type parameters are defined as integers of default kind with the parameter attribute.

The module f90gl provides access to the core OpenGL library procedures, defined constants, and kind type parameters. It may also provide access to one or more OpenGL extensions, along with the related defined constants and derived types.

Additional modules provide access to related libraries, and are given a descriptive name beginning with f90. For example, module f90glu contains the procedures, defined constants and derived types for the OpenGL Utility Library (GLU).

2.2 Types

2.2.1 Numeric

The correspondence between fortran and C numeric types is achieved through use of kind type parameters. The module f90gl_kinds contains the definition of these parameters such that the C representation of an entity of a given OpenGL type agrees with the fortran representation of an entity of the corresponding type and kind whenever possible. When the corresponding representation is not provided by the fortran processor, the lack of said representation remains transparent to the user.

The OpenGL numeric types and the corresponding fortran 90 type(kind) are:

GLbyte	`integer(f90glbyte)`
GLubyte	`integer(f90glubyte)`
GLshort	`integer(f90glshort)`
GLushort	`integer(f90glushort)`
GLint	`integer(f90glint)`
GLuint	`integer(f90gluint)`
GLenum	`integer(f90glenum)`
GLbitfield	`integer(f90glbitfield)`
GLsizei	`integer(f90glsizei)`
GLfloat	`real(f90glfloat)`
GLclampf	`real(f90glclampf)`
GLdouble	`real(f90gldouble)`
GLclampd	`real(f90glclampd)`

The user's code should always specify the kind parameter for all actual arguments passed to OpenGL procedures to insure correspondence between C and fortran types and portability of the user's code:

- Variables should have the kind parameter in the declaration
- Constants should have the kind parameter attached (e.g., `1.0_f90glfloat`)

- Expressions should evaluate to a value with the appropriate kind parameter

The fortran standard does not specify what kinds are to be provided for each type. It is possible that some OpenGL types do not have a corresponding type(kind) on a given fortran processor. On current systems this is highly unlikely for the float, double and long integer types, but may occur with the short integer types. In this case, the implementation of the interface will match fortran and C types in a manner that is transparent to the user. There are at least two approaches that can be taken for this. In the first approach the interface accesses the OpenGL library routine that accepts the available type, rather than the type expected according to the procedure name. In the second approach the C procedure that is called by the fortran procedure converts the arguments to the type specified by the OpenGL definition. If there are any return values of the missing type, they are converted to the available type before returning to the f90gl procedure.

For example, suppose GLshort is a 2-byte integer, GLint is a 4-byte integer, and the fortran compiler supports 4-byte integers but not 2-byte integers, and assume the fortran 90 interface is implemented by a set of "wrapper" functions. Then `f90glshort` will be set to the same value as `f90glint`, which is the kind parameter such that `integer(f90glint)` is a 4-byte integer. Consider an invocation of `f90glVertex2s`. In the first approach, the wrapper function simply invokes `glVertex2i`. In the second approach, the C procedure invoked by the f90gl procedure will accept an argument of type GLint, convert it to type GLshort, and invoke `glVertex2s`.

For the user's application code, this is all transparent. The user declares the argument to be of type `integer(f90glshort)`. If the equivalent of a GLshort is supported by the fortran processor, then the short integer is used; if not, then the equivalent of GLint is used with one of the above methods for handling mismatched type. The user's code works in both environments unchanged.

Note that the equivalent of GLbyte (probably a 1-byte integer) may be supported by the fortran processor, may require promotion to the kind `f90glshort`, or may require promotion to the kind `f90glint` depending on what kinds of integers are supported by the fortran processor.

2.2.2 Logical

The OpenGL logical type and the corresponding fortran 90 type(kind) is:

GLboolean `logical(f90glboolean)`

The type GLboolean is typically a 1-byte entity with the value 0 representing false and nonzero representing true. The fortran processor may or may not support a 1-byte logical type. The kind parameter `f90glboolean`, defined in module f90glkinds, is normally set to the kind parameter for a 1-byte logical if it is supported, or the default kind parameter for logicals if it is not. If the 1-byte logical is not supported, or the fortran representation of logical values does not correspond to the C representation, then the interface routines will perform appropriate type conversions similar to the type conversions described in the section on numeric types.

2.2.3 Character

Some procedures in related libraries and toolkits have character string arguments. These cause no problem in the fortran 90 interface; the dummy argument is given the type `character(len=*)`.

OpenGL functions that return a character string are also no problem in fortran 90. In C the resulting string can be arbitrarily long. In fortran, this is obtained by declaring the function result to be a pointer to an array of type `character(len=1)`, and allocating the pointer inside the function. The user can obtain the number of characters using the `size` intrinsic function, and, if the result is assigned to a pointer variable, can deallocate the memory.

2.2.4 Pointer

Some OpenGL procedures, or procedures in related libraries and toolkits, may require the user to maintain the value of a C pointer. Fortran does not provide pointers in this sense, so this use of pointers is restricted to obtaining a C pointer from an OpenGL procedure, and passing it to another procedure as an actual argument. Thus what is required is a means of storing the bit patterns contained in C pointer variables. The user may also copy a C pointer from one variable to another, which precludes the use of numeric types which are allowed to change the representation (for example, by normalizing the exponent). In the fortran 90 interface a sufficiently long character string is used to store the C pointer one byte at a time. The required length is set in `f90glcptr` in module f90gl_kinds. This is typically 4 and 8 for 32-bit and 64-bit addressing schemes, respectively. Thus a variable of type "C pointer" is declared with `character(f90glcptr)`. This is guaranteed to place the bytes in contiguous "character storage units", which are one byte units for the default character set on all known current fortran processors. The bytes are stored in an order that makes the character string useful as a C pointer.

Some applications require that a C pointer be compared to NULL, thus a null pointer value must be provided. This is defined in module f90gl_kinds as
`character(f90glcptr), parameter :: f90glnullptr = char(0)//...//char(0)`
where the number of `char(0)` is equal to `f90glcptr`, the number of bytes to store a C pointer.

2.2.5 Structures

Some related libraries and toolkits define structures that are used as arguments to the procedures. The fortran 90 interface defines derived types corresponding to these structures. The name of the derived type is obtained from the name of the structure, subject to the same name modification rules used for the fortran 90 procedure names in section 2.3. The derived type definitions are contained in the module for the given library or toolkit. The components of the derived type contain whatever information is required to fulfill the specification of the procedures that operate on that type. Components that may be useful to the user are `public`, but other components may be `private`. An example of where the components are useful to the user is provided by the tk toolkit where a tk procedure sets the components of a derived type, and a GLU procedure needs the values in the components:

```
type (f90tk_rgbimagerec), pointer ::  image
image => f90tkRGBImageLoad( TABLE_TEXTURE )
err = f90gluBuild2DMipmaps(GL_TEXTURE_2D, 3_f90glint, image%sizeX, &
    image%sizeY, GL_RGB, GL_UNSIGNED_BYTE, image%data)
```

For functions that return a C pointer to the struct, the fortran 90 function returns a fortran pointer of the derived type. If the C pointer is NULL, then the fortran pointer is nullified (disassociated), so that the C test "if (cptr == NULL)" is achieved in fortran with "if (.not. associated(fptr))", where cptr and fptr are pointer variables in C and fortran, respectively.

For example, consider the GLU type gluQuadricObj. The fortran 90 type

```
type f90gluquadricobj
  character(f90glcptr) ::  addr
  !  there may be other components, which may be private
end type f90gluquadricobj
```

is defined in module f90glu. The function `f90glunewquadric` would have the effect of

```
function f90glunewquadric()
type(f90gluquadricobj), pointer ::  f90glunewquadric
allocate(f90glunewquadric)
f90glunewquadric%addr = gluNewQuadric()
if (f90glunewquadric%addr == f90glnullptr) then
  deallocate(f90glunewquadric)
  nullify(f90glunewquadric)
endif
end function f90glunewquadric
```

2.2.6 Void

Many OpenGL procedures use the type GLvoid for an argument that may be one of several different types. Generic interfaces provide this capability in fortran 90. Procedures with a GLvoid argument have a generic interface (with the usual name for the procedure as defined in section 2.3) to a set of specific routines, one for each type specified by the OpenGL definition. Additionally, it interfaces to a specific routine that accepts an argument of type `character(f90glcptr)` to allow the GLvoid argument to be a C pointer returned by a prior call to an OpenGL procedure.

Processors that do not support the short integers require additional work here, but it remains transparent to the user. Consider the situation where the fortran processor does not support the kind of integer that corresponds to GLshort. Then `f90glshort` is the same as `f90glint`, so there is no specific routine for the type `integer(f90glshort)`. If the user passes an argument

of type `integer(f90glshort)`, then the specific routine that is called is the one with dummy argument of type `integer(f90glint)`. But, in all such core OpenGL and GLU routines there is another argument that tells what type the GLvoid argument is to be interpreted as. If that argument indicates that the user is passing a GL_SHORT, but the specific routine for `integer(f90glint)` is called because `f90glshort` is the same as `f90glint`, then the interface will handle the mismatched types as described in section 2.2.1. The situation is similar for `f90glbyte`, except that `f90glbyte` could be either `f90glshort` or `f90glint`, depending on the fortran processor.

2.3 Procedures

All OpenGL procedures are available in the fortran 90 interface. The argument lists and return values are identical, subject to the equivalences described in section 2.2. C functions of type void are fortran subroutines; C functions of other types are fortran functions of the corresponding type.

The procedure names in the fortran 90 interface are derived from the C names as follows:

- The name is prepended with `f90`. This insures there are no name space conflicts with either the C library routines or the fortran 77 interface.
- Case is insignificant. This conforms to the fortran 90 requirement that lower case letters are equivalent to the corresponding upper case letters except in a character context.
- Any names that are longer than 31 characters after prepending with `f90` are truncated to 31 characters. This conforms to the fortran 90 requirement that the maximum length of a name is 31 characters. There are no names that require truncating in the core OpenGL and GLU libraries.

2.4 Defined constants

All OpenGL defined constants are provided in module f90gl as integers with the appropriate kind, the parameter attribute, and the same value as in the C interface.

The names for the fortran 90 symbolic constants (parameters) are derived from the OpenGL defined constants as follows:

- Case is insignificant.
- Any names that are longer than 31 characters are truncated to 31 characters. There are no names that require truncating in the core OpenGL and GLU libraries.
- Any names that are not unique after discarding case are replaced with a suitable descriptive name. Specifically, the tk toolkit contains lower case key constants, TK_a through TK_z, and upper case key constants, TK_A through TK_Z. In module f90tk the lower case key constants are named TK_LC_A through TK_LC_Z, with LC standing for lower case. There are no case dependent defined constants in the core OpenGL and GLU libraries.

Note that the names are not prepended with f90 because the symbolic constants are module variables and there is no possibility of name space clashes.

2.5 Dummy procedures

Some routines in related libraries and toolkits take a procedure as an argument. These are declared with the external attribute in the explicit interface provided with the fortran 90 interface. While it is considered by many to be more desirable to provide a complete interface block for dummy procedures, this is not always possible because in some cases there is more than one valid interface for the actual argument.

When the argument is used as a callback function, the procedure may allow NULL as the value of the argument to indicate that the corresponding callback is to be disabled. For example, GLUT uses this technique. When this is the case, the fortran 90 interface for this library provides an external procedure by the name *library-prefix*`nullfunc` which can be passed in place of NULL. For example, the fortran 90 interface to GLUT provides the function `f90glutnullfunc`. Each library requires its own nullfunc procedure in order to preserve the independence of the modules corresponding to each library.

2.6 Array arguments

The explicit interfaces of the fortran 90 interface declare array arguments to be assumed-size arrays, i.e., declared with `dimension(*)`. They are not assumed-shape arrays, declared with `dimension(:)`, because most fortran 90 processors pass assumed-shape arrays as dope vectors containing the dimensions of the array in addition to the starting address. The wrappers would thus be more complicated, to extract the address from the dope vector, and less portable since there is no standard for the dope vectors. There is no loss of functionality by using assumed-size arrays.

3 Implementation

In the fortran 77 binding, the user calls C functions from the fortran program, leading to portability issues and the requirement for the binding to address the interfacing of fortran and C procedures. The fortran 90 interface to OpenGL does not address this issue. The user interface is entirely on the fortran side of the fortran/C interface, therefore the fortran/C interface is contained entirely inside the fortran 90 interface to OpenGL. It is anticipated that most vendor implementations will be for a specific system with specific fortran and C compilers. The containment of the fortran/C interface leaves these implementors free to use whatever system dependent techniques are required for the fortran/C interface without affecting the interface to the user application code. In the case of an implementor attempting to provide an implementation that is portable over several fortran/C/OS combinations, it is left to the implementor to determine how to achieve portability, however the reference implementation may be a useful guideline.

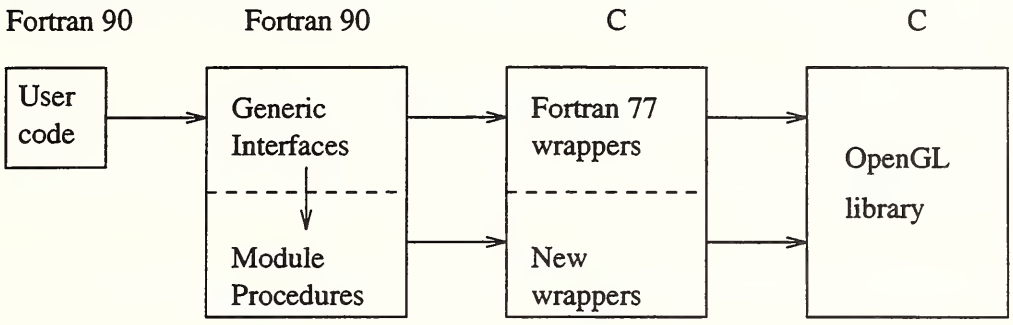

Figure 1: Example implementation using wrappers.

There is no requirement on the actual architectural design of the fortran 90 interface to OpenGL. The only requirement is that the aforementioned modules be provided, and that they provide access to the kind type parameters, procedures, symbolic constants, and derived types described above. However it is anticipated that most implementations will simply provide "wrapper" functions on top of an existing OpenGL implementation. Here the wrapper functions would most likely fall on both the fortran and C sides of the interface. An example of how this might be implemented is illustrated in figure 1.

In this approach, the fortran 90 names for all the OpenGL procedures are defined in generic interfaces in module f90gl. Some of them are used simply to rename the existing fortran 77 interface. Other generic interfaces may include interfaces to module procedures which call new wrapper functions. In particular, this would be used when type conversions are used because the fortran processor does not support the requested type or kind, when one of the arguments is of type GLvoid with several valid types for that argument, or when one of the arguments is a derived type.

In this example, module f90gl would also contain the definition of all the symbolic constants as integers with the parameter attribute and would also use module f90gl_kinds, which makes the kind parameters available to any program unit that uses f90gl.

4 Potential problems

4.1 Assumptions on compilers

The fortran 90 interface to OpenGL is considerably more robust and portable than the fortran 77 interface, however until there is a standard for inter-language calling sequences, it must be assumed that the compilers provide a sufficient inter-language calling convention. Most fortran 90 and C compilers satisfy the following conventions, which are sufficient:

- the kinds of numeric types supported by the fortran processor contain at least the types that the C compiler uses for GLint, GLfloat and GLdouble.

- the numeric types that the fortran and C compilers have in common have the same internal machine representations.
- the fortran and C compilers use the same default character set.
- the fortran processor passes numeric arguments "by reference", i.e., such that the C procedure receives a pointer.
- the fortran processor passes procedure arguments by passing the starting address of the procedure, i.e., such that the C procedure receives a pointer to a function.
- the fortran and C compilers use the same mechanism for transferring arguments between routines, for example pushing them on a runtime stack in the order they appear in the argument list.

Some other assumptions on the compilers can be avoided by using type conversions on both sides of the interface when problems exist:

- assumptions on character string arguments can be avoided by converting the character string to an array of `integer(f90glint)` (or `f90glbyte` if the kind is equivalent to GLbyte) on the fortran side, and back to a character string on the C side.
- assumptions on the existence of a 1-byte logical in the fortran processor can be avoided by converting the logical to an `integer(f90glint)` on the fortran side, and to a GLboolean on the C side.

4.2 Expression actual arguments

Some OpenGL and GLU procedures internally set a pointer to one of the arguments so that the argument can be used by a different procedure called later. In this case it is important that the actual argument not be an expression, which will generate a temporary variable that will no longer exist after returning from the called procedure. Note that array sections and constants are expressions in this context. The user should be warned of this situation. The OpenGL core and GLU procedures effected by this are glBitmap, glFeedbackBuffer, glSelectBuffer, gluNurbsCurve, gluNurbsSurface, gluPwlCurve, and gluTessVertex.

4.3 Unsigned int

Fortran does not provide unsigned integer types; signed integers of the same size are used for these types. The fortran intrinsic function `ibset` can be used for setting values in which the leading bit is a 1. For example, the hexadecimal pattern 8000000A can be set in either an assignment statement or an initialization expression using `ibset` as follows:

```
integer(f90gluint) ::  u = ibset(10,31) !  use bit pattern for 10 and set 31st bit
```

Unsigned integers can also be set in a `data` statement using BOZ notation:

```
integer(f90gluint) u
data / u / z'8000000A'
```

4.4 Array order

The user should remember that fortran stores multidimensional arrays in column major order, whereas C stores them in row major order. Some multidimensional fortran arrays may require transposition. The exception is the transformation matrices passed to glLoadMatrix and glMultMatrix which, as a 4x4 array, are assumed to be in column major order.

5 System installation

The location of the software for the fortran 90 interface to OpenGL is system dependent. The OpenGL documentation provides this information for the user.

5.1 Libraries

The fortran 90 interface procedures may be placed in either the same libraries as the OpenGL procedures (libGL, libGLU, etc.) or in separate libraries (libf90GL, libf90GLU, etc.).

5.2 Module files

Many fortran 90 compilers generate a file containing module information. The name of the file is usually the module name followed by a compiler dependent suffix, for example f90gl.mod. If the compiler generates module files, these are located in the same directory as the OpenGL include files (e.g., gl.h). Some fortran 90 compilers provide a command line option for specifying the location of module files (e.g., -I); with other compilers the module files will have to be copied (or linked) to the user's source code directory.

6 Reference implementation

A reference implementation of the fortran 90 interface for OpenGL is available in the software package called f90gl available from http://math.nist.gov/f90gl. Version 1.0 of the reference implementation covers the OpenGL 1.0 core, GLU, tk, GLUT, and some extensions. A future release will extend this to OpenGL 1.1.

References

[1] Allen Akin, *OpenGL FORTRAN Binding Proposal*, http://www.sgi.com/Technology/openGL/fortran.html

[2] ANSI, *American National Standard for Programming Language – Fortran – Extended*, ANSI, New York, 1992.

[3] Jackie Neider, Tom Davis and Mason Woo, *OpenGL Programming Guide*, Addison-Wesley, 1993.